AMENDMENTS TO THE SPECIFICATION

Please amend the paragraph beginning on line 18 of page 3 and ending on line 10 of page 4 as follows:

This base of information grows exponentially as plugin modules are added and evolve over the life cycle of the software product. Indeed, sweeping upgrades to existing modules may be required each time a new individual plugin is defined. For example, in conventional client/server based systems, the management of **[[a]]**distributed state information for the system requires storage of per-client state information at the service end so that services can rely on certain facts about the client state. Such a server would need to store overhead information about the particular programs and versions available at a client to permit proper interaction between the server and client. Hence, the conventional system for implementing extension creates the significant problem of an ever-growing base of information that is required to maintain compatibility. In conventional systems, each new software extension, or "plugin" module, creates a new set of module interdependencies that must be maintained in all modules that expect to use the newly created plugin. Furthermore, the controlling system must contain all information about the management of the plugin prior to actually loading it and providing its services to the software system. In addition, with conventional plugin modules, user intervention is typically required to effectuate software extension.

Please amend the paragraph beginning on line 5 of page 6 and ending on line 19 of page 6 as follows:

The first plugin module includes an introspection interface portion and a program behavior portion, and may also include an installation program component. The harness evaluates the introspection interface portion of the first plugin module to determine if any resources are required for use of the program behavior portion of the first plugin module. In addition, the computer system may also include a second computer connected to the first computer, where the second computer includes a first resource required by the first plugin module that is transferred to the first computer upon request by the plugin installation program component, or **[[up]]upon** request by the harness. In a selected embodiment, the first resource is a second plugin module that itself includes an

introspection interface portion, an installation manager portion and a program behavior portion. In addition or alternatively, the harness may be configured to evaluate the introspection interface portion of the second plugin module to determine if any resources are required for use of the program behavior portion of the second plugin module. As will be appreciated, either or both of the introspection interface portion and the installation manager portion may be an executable script (e.g., a Tcl script), a command series or an object-code module.

Please amend the paragraph beginning on line 4 of page 7 and ending on line 18 of page 7 as follows:

In accordance with a still further embodiment of the present invention, a method is provided for preprocessing a software module that has an interface portion and a program portion. The method includes receiving a first software module, querying the interface portion of the first software module to identify any resources required by the program portion, and invoking the installation manager portion of the module if all resources required by the program portion are available. For example, the first software module may be received by connecting to a remote computer over a telecommunication network and downloading the first software module from the remote computer. In addition, resources required by the program portion are retrieved that were identified as a result of querying the introspection interface portion prior to invocation of the first software module or as a result of invoking the installation manager portion. In a selected embodiment where the retrieved resource is a second software module comprised of an introspective interface portion, an installation manager portion and a program portion, the interface portion of the second software module **is queried** to identify any resources required by the program portion of the second software module.

Please amend the paragraphs beginning on line 19 of page 9 and ending on line 19 of page 10 as follows:

In accordance with the present invention, the client/server network 100 shown in Fig. 1 may include one or more **[[clients]]client** systems 97, 99, 101 and/or one or more server systems 115, 117 in which plugin manager agents of the present invention may be used. Examples of client/server platforms that may be used with the present invention

include Digital Open VMS, Digital UNIX, DOS, Java, Macintosh, Linux, Microsoft Windows, Sun Solaris, IBM AIX, etc.

Figure 2 shows the plugin manager agent having an introspection interface used in connection with the present invention. As shown in Figure 2, the plugin manager agent encapsulates the details of the interactions between the plugin and the running program that contains the external controller. The controller can extract any of this information that the controller requires by interrogating the **[[introspective]]introspection** interface 135a of the agent. The controller can use the extracted information in order to determine whether to load the plugin component 135b that is encapsulated within the agent 135. The controller can also use extracted information to perform the functions that are necessary to prepare the controller or its containing program to receive the plugin component. Alternatively or in addition, the controller can invoke the agent's installation manager component 135c to cause the load to take place under the control of logic specific to the requirements of plugin component 135b. As the plugin functionality 135b evolves as described above, only a single base of information, that which is contained within the very module that has changed, requires modification. Using the introspective mechanism 135a and the installation manager component 135c, this change and the associated dependency changes are taken into account upon installation and/or invocation in accordance with the present invention.

Please amend the paragraph beginning on line 9 of page 11 and ending on line 20 of page 11 as follows:

In one embodiment, the plugin component 135b is object code, etc., and the installation interface 135a and installation manager 135c are command scripts that encapsulate the plugin component. These commands provide certain interface features, including an introspection capability that **[[respond]]responds** to interrogation with the details of the identity and characteristics of the plugin modules. These characteristics specify the types and versions of other modules or programs that are required for the plugin component to function. The introspection capability may identify not only the other modules required to run the plugin component, but may also identify alternative groupings of modules as options for running the plugin component. The introspection

interface 135a may also provide version information for the plugin to the outside world. The installation manager component 135c of the agent contains executable functionality that implements the logic which is required to correctly load the plugin component into the client system.

Please amend the paragraphs beginning on line 6 of page 12 and ending on line 10 of page 12 as follows:

As yet another alternative embodiment, the plugin functionality 135b and the introspection interface 135a and installation manager 135c are all implemented in **[[a]]** command scripts.

As may be appreciated, the plugin functionality of 135b **[[maybe]]**__**may be**__ of arbitrary form where the interface 135a defines specific details about its installation and use.

Please amend the paragraph beginning on line 13 of page 15 and ending on line 5 of page 16 as follows:

Once the plugin engine has collected the information it needs in order to prepare to load the plugin module 135b, the plugin engine invokes the agent's installation manager 135c to complete the installation. The plugin installation **[[manager135c]]**__**manager 135c**__ is expected to manage the details of loading from this point on, although it may certainly make use of features of the client program. As mentioned above, this can be a multi-step process, whereby a plugin is loaded, decides that it needs to install another plugin in the client program in order to complete its install, and does so using the methods and techniques of the present invention. The installation manager 135c may also locate, transport and store files, program components and other plugins under its own direction and with or without the direct involvement of the plugin engine. This ability is not limited to the actions that are provided by the plugin engine, but may use operating system facilities as well as facilities of other plugin components. The installation manager 135c may also determine and ensure the presence of prerequisite resources through mechanisms that have complete, some or no dependence

on the plugin engine. The installation manager 135c is in this respect an independent agent that is loaded and started by the plugin engine.

Please amend the paragraph beginning on line 18 of page 24 and ending on line 4 of page 25 as follows:

Referring to Fig. 5, the loader 540 interrogates the plugin manager agent 500 for a specification of the API version in use, and if the version of the plugin is supported by the loader, then the plugin manager agent 500 is loaded. However, if the plugin version is not supported by the loader 540 at the client, an alternate loader is obtained. For example, the plugin loader is implemented using an application namespace feature such that multiple, incompatible versions of the loader can co-exist in **[[the in]]** alternate execution namespaces. The client program has at least one version of the plugin loader installed, so other loader versions can be loaded as plugins. All loader versions are available in forms compatible with all other loader versions, so the loader version required in order to load a plugin can always be loaded for use when needed.

Please amend the paragraph beginning on line 11 of page 25 and ending on line 16 of page 25 as follows:

In accordance with the present invention, a user desiring to extend software residing within the client system 101 requests a plugin manager agent module from the server computing system 117. The server computing system 117 has a stored copy of the previously generated plugin manager **[[agent A]]**<u>**agent. A**</u> plugin delivery module 123 forwards the plugin manager agent to the client system 101. The plugin manager agent 135 is received by a communication device 107 and stored in the memory 105 of the client 101.

Please amend the paragraphs beginning on line 14 of page 26 and ending on line 8 of page 27 as follows:

FIG. 7 is an exemplary embodiment of a client system within the client/server network 100 of FIG. 1, illustrating controller installation of plugin manager agent modules using a controller 201 to enable dynamic extensibility of coded applications.

More specifically, a controller 201 containing a interface processing engine 203 installs plugin modules 233, **[[235]]238** and 239 to extend applications 205, 235 as required.

To initiate processing, the controller 201 downloads plugin manager agent module 233, either from a server, through a network or from some other storage media. Residing within the client system 200 are applications 205 and 235. The application 205 includes several modules 237, 209, 211 and 215 that require extensibility. On receiving the plugin manager agent module 233, the controller 201 invokes the introspective interface portions 233a of script **[[plugins]]plugin** module 233 using the interface processing engine 203. As the interface logic is evaluated, all system dependencies and versioning information are correlated to determine if the client system 200 has the resources required by module 233. Once all the required criteria have been met, the controller 201 completely installs plugin module 233 by invoking the plugin manager agent's installation manager component. If additional modules (e.g., modules **[[235]]238**, 239) are required to run module 233, the controller 201 accesses a server to retrieve the modules **[[235]]238**, 239, when such a plugin is available.